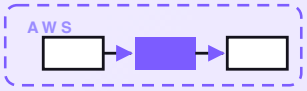
7-PART SERIES · FREE COMPANION



Survey analyzer

A serverless analyzer that makes sense of open-ended survey answers. It reads every free-text response, groups them into the handful of themes people actually raised, counts how common each theme is, pulls a representative quote for each, and emails the owner a short “here’s what customers are telling you” summary. Anything urgent — an angry customer, a threat to leave — gets flagged for a human right away. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Survey analyzer

PART 1 OF 7

MAY 15, 2026 PART 1 OF 7 · SURVEY ANALYZER SERIES ~5 MIN READ

A survey analyzer on AWS for a few dollars a month

A small business runs a survey and gets back a pile of free-text answers. The “anything else you’d like to tell us?” box, filled in by three hundred people. Most owners scroll the first twenty, feel like they’ve got the gist, and never read the rest. The real signal — the same complaint raised forty different ways, the one furious customer buried on page nine — gets lost in the scroll. Reading every answer is slow, and skimming a few is how you miss the thing that actually matters. This post walks through the design of a small analyzer that reads every answer, groups them into the handful of themes people actually raised, pulls a real quote for each, and emails you a short summary — while flagging anything urgent for a human the moment it lands.

KEY TAKEAWAYS

- Three sources for answers: a form-submit lane, an inbox forwarding lane, and a Drive sheet of exports.
- Every answer gets a quick urgent check the moment it lands; everything else waits for the weekly grouping pass.
- Grouping is embeddings plus plain-code clustering; a model only names each theme and picks one real quote.
- The weekly summary names each theme, counts it, and quotes it — written from the groups, never one answer.
- Designed on AWS for about $3.20/month at typical small-business volume.

The whole system on one page

Before any code, here's the shape of what we're designing.

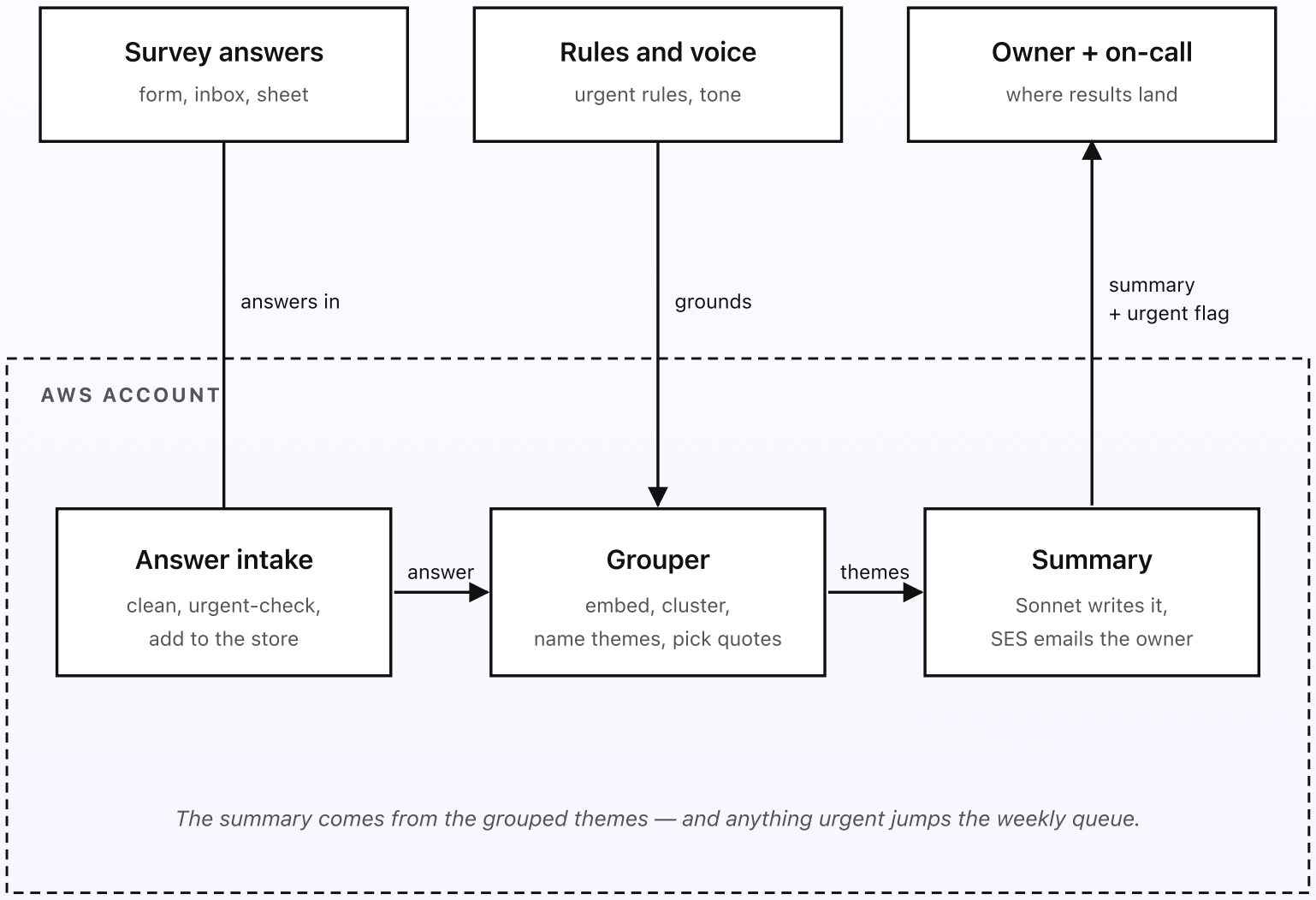

Survey answers
form, inbox, sheet
Rules and voice
urgent rules, tone
Owner + on-call
where results land
answers in
grounds
summary
+ urgent flag
AWS ACCOUNT
Answer intake
clean, urgent-check,
add to the store
answer
Grouper
embed, cluster,
name themes, pick quotes
themes
Summary
Sonnet writes it,
SES emails the owner
The summary comes from the grouped themes — and anything urgent jumps the weekly queue.

Fig 1. Three sources outside, three pieces inside AWS. Answers flow in from a form-submit lane, an inbox forwarding lane, and a Drive sheet. The Grouper runs weekly and turns answers into themes. The Summary piece writes a short email and the urgent lane jumps the queue.

What you set up once (the outside)

- **Survey answers.** The free-text responses to your survey. The backbone lane is the form-submit lane: your survey form posts each response to a small endpoint the moment someone hits send. Two other lanes (covered in Part 2) catch the rest — an inbox-forwarding lane (forward an email reply or an export to a dedicated address and the analyzer takes it from there) and a Drive sheet (paste an export in and a small sync picks it up). Each answer carries a little context if you have it: the survey it came from, the date, and a rating if your survey collected one.
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc says what counts as urgent — an angry customer, a threat to leave, a safety or legal concern, a request for a callback — who gets the urgent flag, and how big a theme has to be before it's worth reporting (so a single one-off comment doesn't become a "theme"). The *voice* doc holds the tone for the summary and the email layout: how warm, how blunt, how long.
- **Owner and on-call.** The owner gets the weekly summary email. The on-call person gets the urgent flag the moment an at-risk answer lands — with the full text and a plain note on why it tripped. In a small shop these are often the same person; the system just lets them be different.

What runs on each answer and each week (the inside)

- **The answer intake.** Three sources feed the store. Each new answer is cleaned (strip signatures, drop blanks, trim), then run through a quick urgent check — a small, cheap model call that asks “does this need a human today?” If yes, it’s flagged right away (Part 5). Either way the cleaned answer is written to the store with its context.
- **The grouper.** Runs once a week. Reads every answer from the period. Turns each one into a vector with Titan embeddings — a vector is just a list of numbers that captures what the answer is about, so answers that mean the same thing land close together even when the words differ. Plain-code clustering then groups the close ones. For each cluster, a Claude Haiku 4.5 call writes a short plain-English name (“Slow checkout”, “Loved the staff”) and picks one real answer that best represents the group. The clustering math is plain Python; the model only labels.
- **The summary.** Reads the named themes, their counts, and their quotes. A Claude Sonnet 4.6 call writes a short summary: the response count, the overall mood, the handful of themes with counts and quotes, and a closing line on anything flagged urgent that week. SES emails it to the owner. The summary is written from the groups, never from one cherry-picked answer.

In plain words

You send a post-visit survey and 280 people reply to the open box. Through the week, each reply lands, gets cleaned, and gets a quick urgent check. On Tuesday, one reply says “third time the order was wrong, I’m done, cancel my account” — the urgent check trips and your on-call person gets it within a minute, full text attached, before it ever shows up in a summary. The other 279 wait. On Sunday

night the grouper runs: it turns every answer into a vector, clusters them, and finds five real themes — “slow checkout” (61 answers), “friendly staff” (44), “parking is hard” (38), “prices crept up” (22), and a long tail. Monday morning you get one short email: the mood, those five themes with counts, one real quote each, and a note that one urgent answer came in Tuesday and was handled. You read the whole picture in ninety seconds instead of skimming twenty replies and guessing.

The cost of running this is about $3.20 a month at SMB volume. The cost of *not* running it is the survey you paid to send and then only half-read — and the angry customer on page nine you never saw until they were already gone.

DESIGN RULES THAT SHAPED EVERY DECISION

- The summary is written from the grouped themes, with counts and real quotes — never from one answer that caught the model's eye.
- Urgent answers jump the queue. An angry customer doesn't wait until Sunday; they go to a human the minute they land.
- The model labels and writes; the grouping math is plain code. A theme's count is a real count, not a guess.
- Every quote in the summary is a real answer somebody actually wrote, shown verbatim, not a paraphrase.
- A theme has to clear a minimum size to be reported, so one stray comment never gets dressed up as a trend.
- Every urgent flag and every weekly run is logged, so you can audit what the analyzer saw and did months later.

Why this shape

Most teams deal with open-ended survey answers in one of three ways: read a handful and call it a sense of the room, run every answer past a person who burns an afternoon tagging them, or never open the export at all. The handful misleads — the first twenty replies are not the other 260. The afternoon of tagging is real work that gets skipped the moment the week gets busy. And the unopened export is the most common outcome of all: you paid to gather feedback and then never actually heard it.

The setup above keeps the answers somewhere the team already has them, but adds a small system that *reads every one*. It uses AI exactly where plain code can't cope — grouping messy free text by meaning — and plain code everywhere else, so the counts are trustworthy and the bill stays tiny. It surfaces the few themes that matter instead of a wall of raw text. And it never makes you wait a week for the one answer that needed you today. The analyzer is quiet on the weeks nothing stands out; it only speaks up with the short version and the thing that can't wait.

The next four posts walk through each piece in turn: how a survey answer gets collected, how answers get grouped into themes, how a summary reaches the owner, and how an urgent answer gets flagged. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 15, 2026 PART 2 OF 7 · SURVEY ANALYZER SERIES ~4 MIN READ

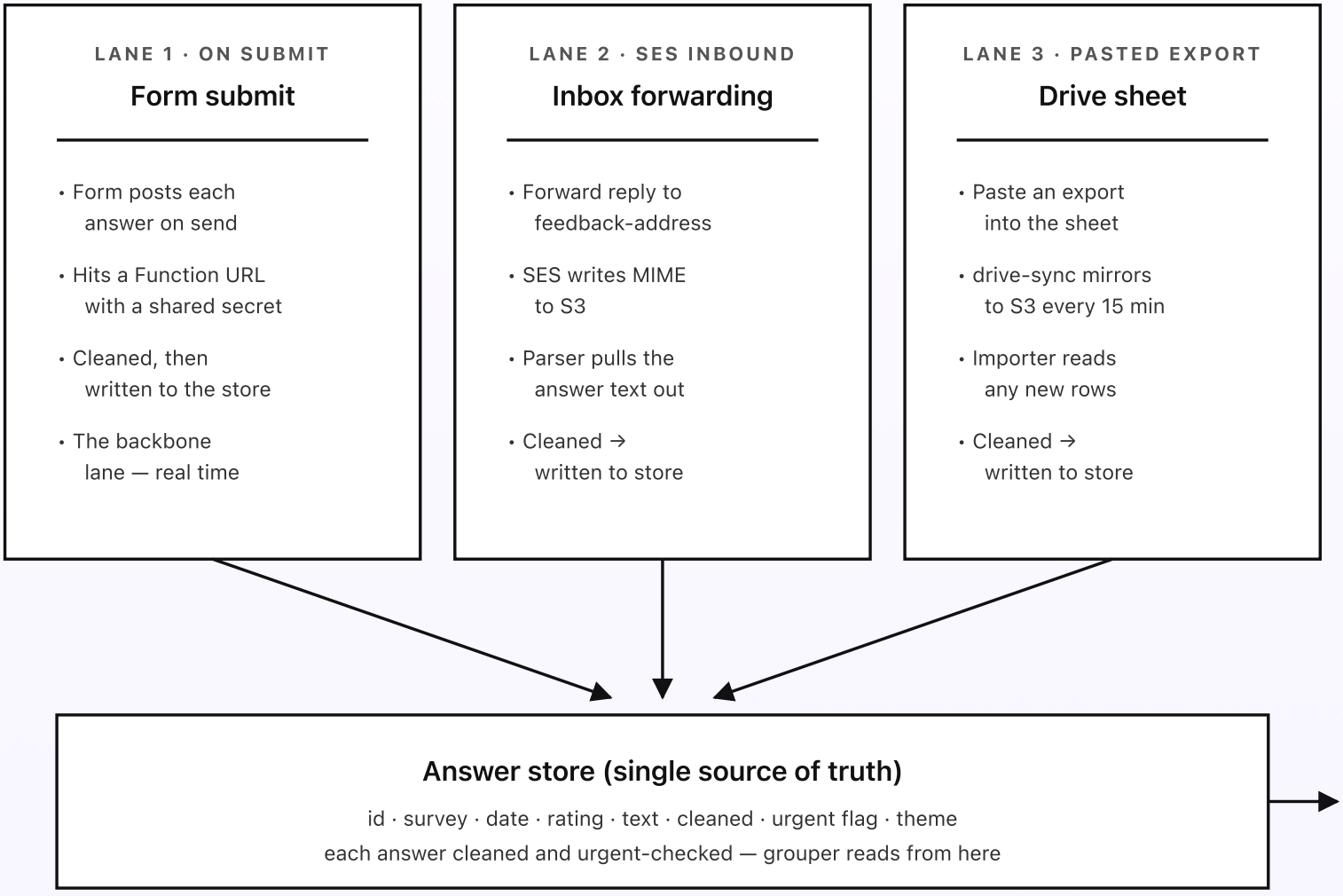
How a survey answer gets collected

The analyzer can only make sense of answers it actually has. So the first job is making sure every free-text response ends up in one place. There are three ways an answer gets in: your survey form posts it to a small endpoint the moment someone hits send, somebody forwards an email reply or an export to a dedicated address, or you paste an export into a Drive sheet. The form lane is the backbone. The other two exist because in real life feedback arrives in a dozen shapes and nobody re-keys it by hand.

KEY TAKEAWAYS

- Three intake lanes feed one store: a form-submit lane, an inbox-forwarding lane, and a Drive sheet.
- The form lane posts each answer to a Lambda Function URL the moment it is submitted.
- Forwarded emails and exports arrive via SES inbound; a parser pulls the answer text out.
- Every answer is cleaned and gets a quick urgent check before it lands in the store.
- The store is the single place the grouper reads from — the other lanes just write into it.

Three lanes into one store



The store is the single source of truth — the other lanes are conveniences that write into it.

Fig 2. Three lanes converge on one answer store. The form lane is the backbone; the inbox lane and the sheet lane are conveniences that write into the same store. Each answer is cleaned and urgent-checked before it settles, and the grouper reads only from the store.

Lane 1: the form-submit lane (the backbone)

The main lane. Most survey tools and web forms can send a copy of each submission to a URL the moment it's filled in. You point that at a small Lambda Function URL — a plain web address that runs a little code, with no API Gateway in front of it. The endpoint checks a shared secret so only your form can post to it, reads the fields you care about (the free-text answer, plus any survey name, date, and rating), cleans the text, runs the quick urgent check from Part 5, and writes the answer to the store in DynamoDB. The whole thing takes a few hundred milliseconds and costs a fraction of a cent.

This lane covers the steady state: someone fills in your survey, and seconds later the answer is in the store, cleaned and checked. Most answers arrive this way.

Lane 2: inbox forwarding (for replies that arrive as email)

Plenty of feedback never goes through a form. A customer replies to your "how did we do?" email with a paragraph. A team member exports last quarter's answers from an old tool and wants them in the same place. Set up a dedicated inbound address — something like `feedback@your-company.com` — via Amazon SES. Anyone forwards a reply or an export to that address and the analyzer takes it from there. SES writes the raw email to S3. That triggers a parser Lambda that walks the email, pulls the answer text out of the body (or out of a simple CSV or

spreadsheet attachment), strips the signature and the quoted “on Monday you wrote” trail, cleans it, and writes one answer per response into the store.

The parser leans on plain code for the common cases — a plain-text body, a CSV column — and only reaches for a small model call when the layout is genuinely ambiguous. Most forwarded feedback is plain enough that no model is needed at all.

Lane 3: the Drive sheet lane

The simplest lane for a bulk paste. You keep a Google Sheet in a Drive folder with one column for the answer text and a few optional columns for survey name, date, and rating. Paste an export in and you’re done. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://sa-answers-source/answers.csv` if the sheet has changed since the last sync. A tiny importer reads any rows it hasn’t seen before, cleans them, runs the urgent check, and writes them to the store. Reading from S3 keeps Drive API calls predictable and gives you S3 versioning for free, so a bad paste can be rolled back in one click.

This lane is the most hands-on of the three, but it’s the one that needs no setup beyond a shared sheet — handy for a team migrating old feedback or running an occasional one-off survey.

Why everything funnels into one store

Three lanes in, but only one place the grouper actually reads. That’s a deliberate constraint. If each lane wrote its answers somewhere different, every “why didn’t

this show up in the summary?" question would mean checking three places and reconciling three formats. Funneling everything through one store means there is exactly one shape for an answer — id, survey, date, rating, raw text, cleaned text, urgent flag, and (once grouped) its theme — and exactly one place to look. The convenience lanes are first-class for getting answers in, but they always land in the same store on the way.

Next post: how the grouper reads the store, turns each answer into a vector, clusters the vectors, and names the themes.

PART 3 OF 7

MAY 15, 2026 PART 3 OF 7 · SURVEY ANALYZER SERIES ~5 MIN READ

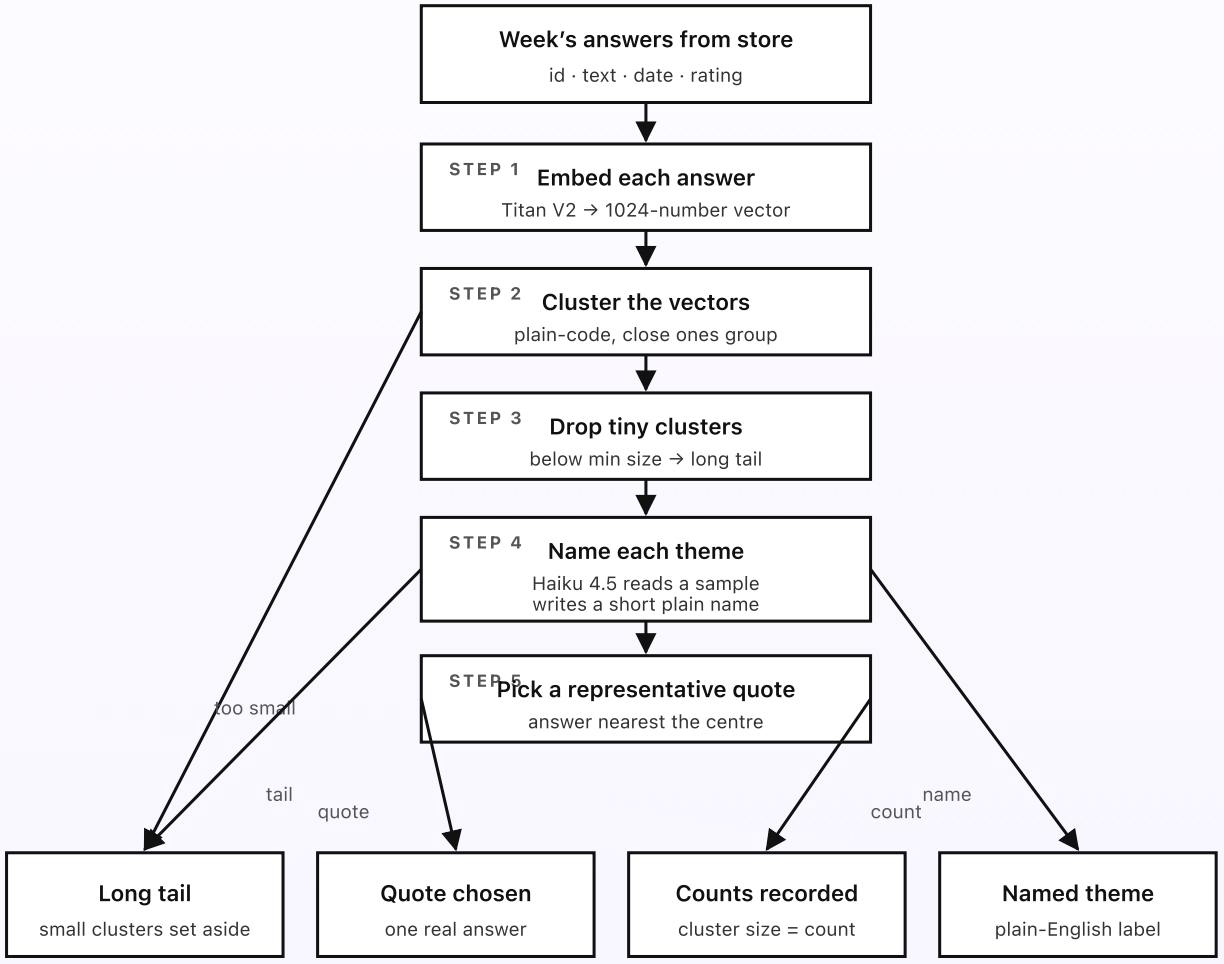
How answers get grouped into themes

Once a week, an EventBridge Scheduler rule fires the grouper Lambda. The Lambda reads the week's answers, turns each one into a vector, groups the close ones together, and lands on a handful of real themes — each with a count and one real quote. This is the one place in the whole system where AI genuinely earns its keep: plain code can't tell that "took forever to pay" and "the line at the till was endless" are the same complaint. Embeddings can. But the counting is still plain math, and the model only ever names and quotes — it never decides the numbers.

KEY TAKEAWAYS

- The grouper runs once a week via EventBridge Scheduler — not on every answer.
- Each answer becomes a vector via Amazon Titan Text Embeddings V2 (1024 numbers per answer).
- Close vectors are clustered with plain code; the cluster sizes are the theme counts.
- Claude Haiku 4.5 names each cluster and picks one real answer as the representative quote.
- A cluster has to clear a minimum size to count as a theme; small ones fall into a long tail.

The grouping flow, per weekly run



The model only names and quotes — the cluster sizes are the counts, in plain code.

Fig 3. The grouper's flow, per weekly run. Five steps turn a week of raw answers into a handful of named, counted, quoted themes. The model labels and quotes; the clustering and the counting are plain code.

Step 1: turn each answer into a vector

The grouper reads the week's cleaned answers from the store. For each one, it calls Amazon Titan Text Embeddings V2 and gets back a vector — a list of 1024 numbers that captures what the answer is *about*. The useful property is that two answers with the same meaning produce vectors that sit close together, even when they share no words. "Took forever to pay" and "the line at the till was endless" end up near each other; "loved the staff" ends up somewhere else entirely. The vectors are saved in an Amazon S3 Vectors index so the grouper can search and compare them cheaply without running a database server.

This is the step plain code simply cannot do. Keyword matching would put "till" and "pay" in different buckets and miss that they're the same gripe. Embeddings are what make grouping by meaning possible at all — which is exactly why this is one of the few places the system reaches for a model.

Step 2: cluster the close vectors

With every answer now a point in space, grouping the close ones is a plain-math job — no model needed. The grouper runs a standard clustering routine in Python that finds dense groups of nearby points and leaves the scattered ones ungrouped. Answers that mean roughly the same thing fall into the same cluster; genuinely different answers form their own. Crucially, the number of answers in

each cluster is just a count — a real, exact number, computed by counting rows. When the summary later says “61 people raised slow checkout,” that 61 is a count of actual answers, not a model’s estimate.

Step 3: set aside the tiny clusters

Not every cluster is a theme. Three people mentioning the same oddly specific thing is interesting, but it isn’t a trend, and a summary that lists fifteen “themes” of two answers each is just noise in a nicer font. The rules doc sets a `min_theme_size` (default around 1% of the week’s answers, with a floor of five). Any cluster below that is folded into a long tail that the summary mentions in one line (“plus a scattering of one-off comments”) but doesn’t break out. This keeps the summary to the handful of themes that actually matter.

Step 4: name each theme

Now the model earns its second keep. For each surviving cluster, the grouper sends Claude Haiku 4.5 a small sample of the answers in it and asks for a short, plain-English name: “Slow checkout,” “Friendly staff,” “Parking is hard.” The prompt is tight: “Name the shared topic in three or four words. Don’t add a topic that isn’t in these answers. Return the name only.” The model never sees the counts and never decides which answers belong to which group — the clustering already did that. It’s purely putting a readable label on a group plain code already formed.

Step 5: pick one real quote

A count and a name tell you *what* people said and *how many*; a quote tells you how it *felt*. For each theme, the grouper picks the answer sitting closest to the centre of its cluster — the most representative single response — and stores it verbatim as the theme's quote. It's a real thing a real person wrote, shown word for word, never a paraphrase the model invented. If the most central answer is unusually long, Haiku is allowed to trim it to a clean sentence, but only by cutting — never by rewording.

Why weekly, and why this split of work

The grouper runs weekly rather than on every answer for two reasons. First, grouping only makes sense in bulk — you can't cluster one answer. Second, embedding and clustering a whole week at once is far cheaper than re-running the math every time a single answer lands. The urgent lane in Part 5 is what handles the "can't wait" case; the grouper is deliberately the slow, thorough, weekly read.

And the split of work — model for embeddings, naming, and quote-trimming; plain code for clustering and counting — is the whole reason the numbers are trustworthy. The expensive, fuzzy judgment (what does this mean?) goes to the model. The exact, checkable arithmetic (how many?) stays in code where it can't drift. That's what lets the summary say a number and mean it.

Next post: how the summary gets written from these themes and reaches the owner's inbox — and the guardrails that keep it honest.

PART 4 OF 7

MAY 15, 2026 PART 4 OF 7 · SURVEY ANALYZER SERIES ~5 MIN READ

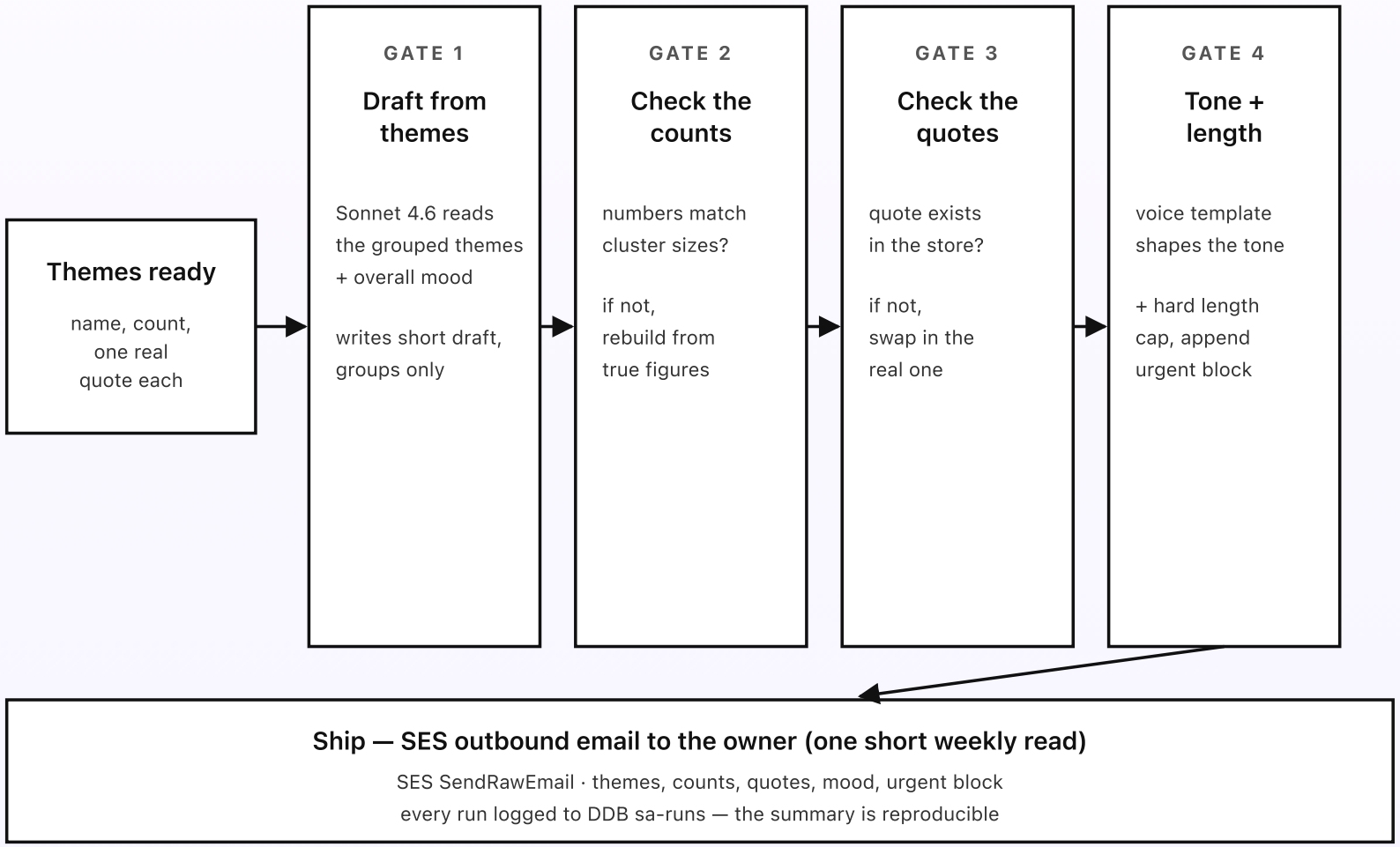
How a summary reaches the owner

The grouper handed off a handful of themes — each with a name, a count, and a real quote. Now the summary Lambda has to turn that into one short email the owner will actually read, and make sure every number and every quote in it is true. Get this wrong and the summary is worse than no summary: an inflated count, a quote nobody wrote, a wall of text nobody finishes. Four small guardrails sit between the grouped themes and the email that lands.

KEY TAKEAWAYS

- Claude Sonnet 4.6 writes the summary from the grouped themes — never from one cherry-picked answer.
- Counts in the email are checked against the real cluster sizes before anything is sent.
- Every quote is verified to be a verbatim answer that actually exists in the store.
- The voice doc sets the tone and a hard length cap so the email stays a one-page read.
- SES emails the owner; the urgent items from the week are appended at the bottom.

Four guardrails on every summary



Every number and every quote is checked against the store — the summary can't drift from the truth.

Fig 4. Four guardrails between the grouped themes and the email. Draft from the groups. Check every count. Check every quote. Shape the tone and length. Then ship via SES and log the run so the summary is reproducible.

Gate 1: draft from the themes, not the answers

The summary Lambda hands Claude Sonnet 4.6 the structured output of the grouper: each theme's name, its exact count, and its one real quote, plus the overall mood (a simple share of positive, neutral, and negative answers, computed in plain code from the ratings and the urgent flags). Sonnet's job is narrow: write a short, warm, plain summary of *this* — a top line on volume and mood, then the themes in order of size, each with its count and quote. The prompt is explicit that it must work only from the supplied themes and may not introduce a topic, a number, or a quote that isn't in the input. It never gets to rummage through raw answers and surface its own favourite.

Sonnet rather than Haiku here is a deliberate choice. This is the one piece of writing the owner actually reads, and it has to weigh several themes, judge what leads, and strike a tone — the kind of light reasoning the heavier model does noticeably better. Everywhere else, the cheap model is fine; here the extra cents buy a summary worth reading.

Gate 2: check every count against the real sizes

A model writing prose around numbers will, now and then, round "61" to "about 60" or merge two themes and add their counts wrong. So before anything sends, Gate 2 pulls every number out of the draft and compares it against the cluster

sizes the grouper recorded. If a count doesn't match exactly, the draft is rejected and rebuilt — the gate hands the true figures back to Sonnet and asks for the summary again, or, after a couple of tries, drops in the numbers itself. The owner never sees a count the system can't stand behind.

Gate 3: check every quote is real

The same care goes to the quotes. Each quoted sentence in the draft is looked up against the store to confirm it's a verbatim answer somebody actually wrote. A quote the model softened, tidied, or invented fails the check and is swapped for the real representative quote the grouper picked in Part 3. This matters more than it sounds: a quote is the one place a summary feels like real customers talking, and a made-up quote — however plausible — quietly turns the whole report into fiction. Every quote that ships is real.

Gate 4: tone, length, and the urgent block

The voice doc holds the tone — how warm, how blunt, how much hedging — and a hard length cap. Gate 4 shapes the verified draft to that voice and trims anything over the cap, because a summary that scrolls is a summary that doesn't get read. Then it appends a short closing block: the answers that were flagged urgent during the week (from Part 5) and a one-line note on how each was routed, so the owner sees in one place both the slow signal (the themes) and the fast one (the urgents). The finished email goes out through SES outbound, and the whole run — inputs, draft, final — is written to DynamoDB so the summary is reproducible and auditable later.

Why the guardrails exist

None of these gates are exotic. They're the care a thoughtful analyst would take if they were writing the summary by hand — quote real people, get the numbers right, keep it short, and put the urgent stuff where the boss will see it. Putting them in code as four sequential checks makes honesty a property of the system, not something you're trusting one model call to remember on a busy Sunday night. The owner can act on the summary precisely because they never have to wonder whether a number or a quote was real.

Next post: the urgent lane — how a single answer that can't wait until Sunday gets pulled out and sent to a human the minute it lands.

PART 5 OF 7

MAY 15, 2026 PART 5 OF 7 · SURVEY ANALYZER SERIES ~5 MIN READ

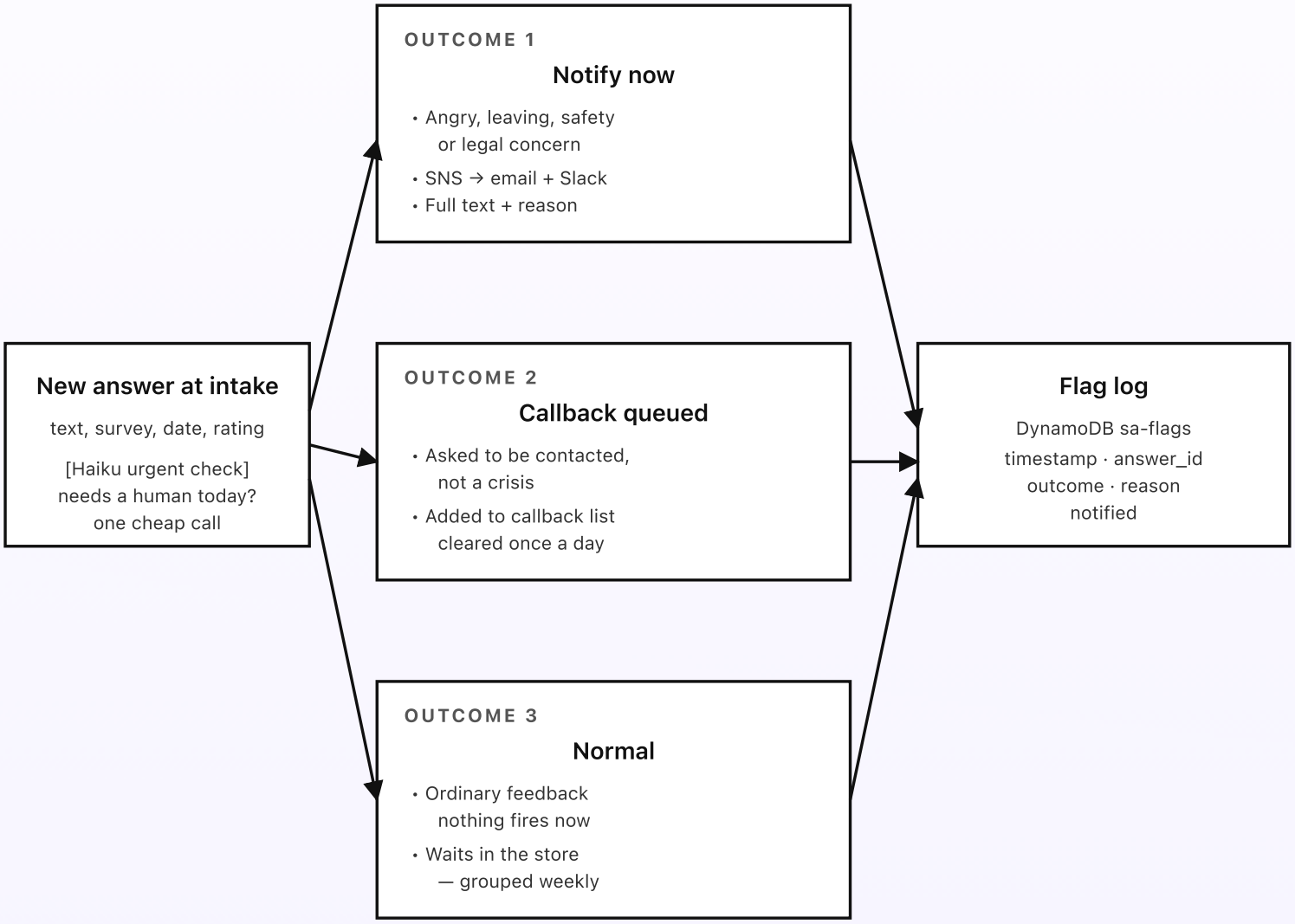
How an urgent answer gets flagged

A weekly summary is the right speed for most feedback. It is exactly the wrong speed for the customer who just wrote “third time the order was wrong, I’m done, cancel my account.” That answer can’t wait five days to show up in a tidy theme. So every answer, the moment it lands, runs a quick urgent check — and anything that trips jumps the whole queue and goes straight to a human. This post walks through what the check looks for, what happens when it fires, and why a real person, not the system, decides what to do next.

KEY TAKEAWAYS

- Every answer runs a quick urgent check at intake — a small, cheap Claude Haiku 4.5 call.
- It looks for an angry customer, a threat to leave, a safety or legal concern, or a callback request.
- If it trips, the answer is sent to a human within a minute via SNS — email or Slack.
- The system only flags and routes; the human decides what to actually do.
- Every flag is logged, and the week's flags are appended to the weekly summary too.

Three outcomes of the urgent check



The system only flags and routes — a human decides what to do. Every flag is logged.

Fig 5. One quick check at intake, three outcomes. Notify now pages a human within a minute. Callback queued holds it for a daily sweep. Normal just waits for the weekly pass. Every outcome writes to the flag log.

The check itself: one small call at intake

When an answer lands — from any of the three lanes in Part 2 — the intake runs a single Claude Haiku 4.5 call before the answer settles in the store. The prompt is short and the rules doc supplies the definition of urgent, so the business can tune it: “Read this survey answer. Does it need a human today? Answer urgent, callback, or normal, and give a one-line reason. Urgent means an angry customer, a threat to cancel or leave, a safety or legal concern, or anything that reads like a complaint that will escalate. Callback means they asked to be contacted but it isn’t a crisis. Otherwise, normal.” It’s Haiku, not Sonnet, because this runs on *every* answer and the judgment is a simple sort, not an essay — the cheap model is both fast enough and good enough.

The check looks only at the one answer in front of it. It doesn’t need the embeddings, the clusters, or the rest of the week — it’s a fast triage at the door, deliberately kept separate from the slow, thorough grouping that happens later.

Outcome 1: notify now

If the answer trips the urgent bar, the analyzer publishes it to an SNS topic immediately. SNS fans the message out to whoever the rules doc named as on-call — an email address, a Slack channel, or both. The message carries the full answer text (not a summary — the human needs the real words), the one-line

reason it tripped, the survey and date it came from, and any rating. The whole path from “answer lands” to “phone buzzes” is well under a minute. The answer is still stored and still grouped into a theme the following week; flagging it urgent doesn’t pull it out of the normal picture, it just adds a fast lane on top.

Crucially, the system stops there. It does not reply to the customer, does not offer a refund, does not promise anything. It hands a real person the words and the reason and gets out of the way. The most expensive mistake a feedback system can make is auto-answering an angry customer with something tone-deaf, so this lane is flag-and-route only — a human owns every word that goes back.

Outcome 2: callback queued

Plenty of answers ask for contact without being a crisis — “could someone call me about my account?”, “happy to chat more if useful.” Paging the on-call person for those would train them to ignore the alerts, which is the worst outcome of all. So a callback answer goes onto a short list that the on-call person clears once a day rather than firing an instant alert. It’s the middle speed: faster than the weekly summary, calmer than a page. The answer is stored and grouped as normal too.

Outcome 3: normal

The common case. Most feedback is ordinary — a compliment, a small gripe, a suggestion — and nothing needs to happen today. The answer simply waits in the store for the weekly grouping pass, where it becomes part of a theme and a count. No alert, no queue, no noise. Keeping this the default is what makes the urgent

lane trustworthy: because the bar is set so that most answers are normal, an alert actually means something when it fires.

Why a human always decides, and why every flag is logged

Two design rules hold this lane together. First, the system never acts on the customer's behalf — it surfaces and routes, and a person decides what to do. That keeps the analyzer firmly in "read-only" territory: it can't send a bad reply because it can't send a reply at all. Second, every outcome — urgent, callback, or normal — writes a row to the `sa-flags` table with the timestamp, the answer id, the outcome, the reason, and who was notified. That log is what lets you tune the rules ("we're getting paged too often — tighten the definition"), prove a complaint was seen the day it arrived, and append the week's flags to the bottom of the summary so the slow and fast lanes meet in one place.

Next post: the cost breakdown. The whole pipeline above — embeddings, a weekly grouping pass, a per-answer urgent check, and one summary — runs in coffee-money territory at SMB volume; Part 6 explains exactly where the dollars go.

PART 6 OF 7

MAY 15, 2026 PART 6 OF 7 · SURVEY ANALYZER SERIES ~3 MIN READ

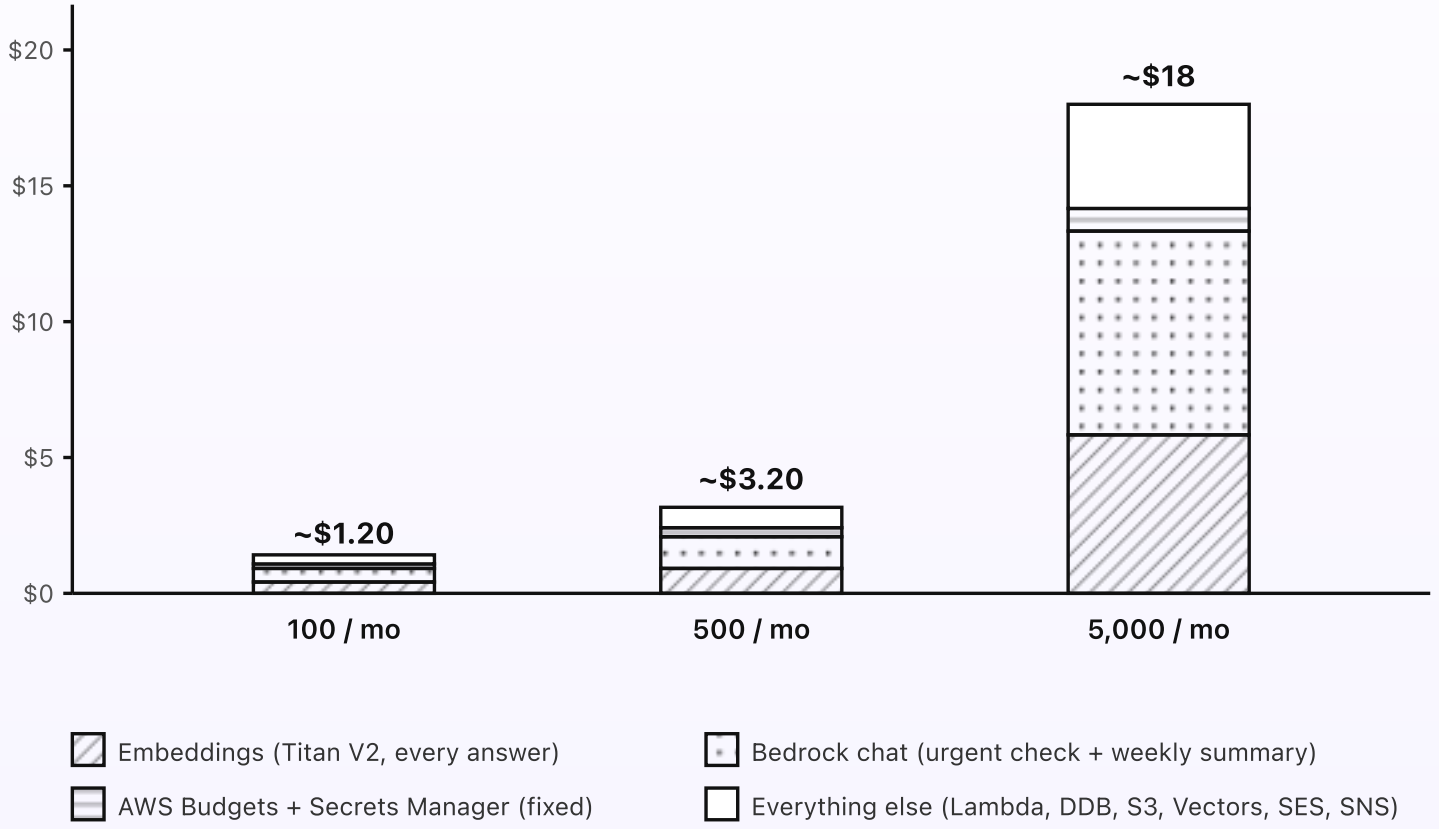
What the survey analyzer costs

The survey analyzer leans on AI more than most systems in this series — it embeds every answer, checks every answer for urgency, and writes one summary — so the bill is a little higher than a pure date-arithmetic system. It is still small. The grouping pass runs once a week, not on every answer. The urgent check uses the cheap model. Only the weekly summary uses the heavier one, and only once. At typical SMB volume, the bill is a few dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $3.20/month at typical SMB volume (around 500 responses a month).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- Bedrock is the biggest slice: embeddings on every answer, plus a cheap urgent check on each.
- The summary is one larger Sonnet call a week — a few cents, not the bulk of the bill.
- At 1,000 responses the bill is around $6. At 5,000 it's around $18.

Cost at three volumes



The per-answer work — embedding and the urgent check — is the dominant cost, and still fractions of a cent each.

Fig 6. Monthly cost at three response volumes. Embeddings and the per-answer urgent check are the biggest slices because both run on every answer. The weekly summary is one larger call. The fixed cost stays near zero.

Where the dollars actually go

Embeddings (a big slice). Every answer is turned into a vector with Amazon Titan Text Embeddings V2. A survey answer is short — usually well under 200 tokens — so each embedding is a fraction of a cent. The cost grows straight in line with response count, because every answer gets one. At 500 responses a month it's well under a dollar; at 5,000 it's a few dollars.

Bedrock chat (the other big slice). Two callsites. The urgent check fires Claude Haiku 4.5 once per answer — a short prompt and a one-word answer, so a tiny fraction of a cent each, but it runs on every response, so it tracks volume like the embeddings do. The weekly summary fires Claude Sonnet 4.6 once a week: it reads the grouped themes and writes a paragraph, a few thousand input tokens and a few hundred output tokens, so a couple of cents per run — about a dime a month. The naming calls inside the grouper (Haiku, one per theme) add a handful of cents.

Lambda runtime. The intake Lambda fires per answer (cheap and brief). The grouper runs once a week and is the heaviest single invocation — it embeds and clusters the whole period — but even at 5,000 answers that's seconds of compute. Add the drive-sync Lambda every fifteen minutes and the summary Lambda once a week, and the Lambda total still lands under a dollar at all three volumes.

S3 + S3 Vectors. The mirrored answer CSV, the raw MIME from forwarded feedback, and the vector index. A few MB at SMB volume. S3 Vectors is priced for exactly this — storing and searching a modest set of embeddings without running a database — so it stays in the cents.

DynamoDB on-demand. The answer store plus the `sa-flags` and `sa-runs` tables. Writes on intake, reads on the weekly pass. Pennies a month at any of these volumes.

SES and SNS. SES sends one summary email a week and the occasional forwarded-feedback receipt: a couple of cents a year. SNS fans out the urgent flags: $0.50 per million publishes, so effectively free at a handful a week. Both negligible.

What doesn't cost money

- **API Gateway.** Replaced by a Lambda Function URL for the form-submit endpoint.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The grouper runs once a week and the intake only when an answer lands.
- **A managed vector database.** S3 Vectors stores the embeddings; there's no always-on search cluster to pay for.
- **Sonnet on every answer.** The heavy model writes one summary a week. Per-answer work uses Haiku and Titan, the cheap paths.

How the cost scales

Embeddings and the urgent check grow straight in line with response count, because both run on every answer. The weekly summary and the theme-naming calls barely move — they're tied to the number of themes, not the number of

answers. So the bill at 10,000 responses a month is around $35; at 25,000 it's around $80. Past those volumes you'd batch the embeddings and maybe move the urgent check to a cheaper keyword pre-filter that only escalates borderline answers to the model — optimizations, not redesigns.

Set an AWS Budgets alarm at $30/month so anything unusual pages you before the bill matters. The analyzer's normal-volume bill stays well under that ceiling.

Last post in the series: the engineering reference. Same system, drawn for engineers — service names, Lambda inventory, IAM scopes, the S3 Vectors index, DynamoDB schemas, and EventBridge Scheduler config.

PART 7 OF 7

MAY 15, 2026 PART 7 OF 7 · SURVEY ANALYZER SERIES ~8 MIN READ

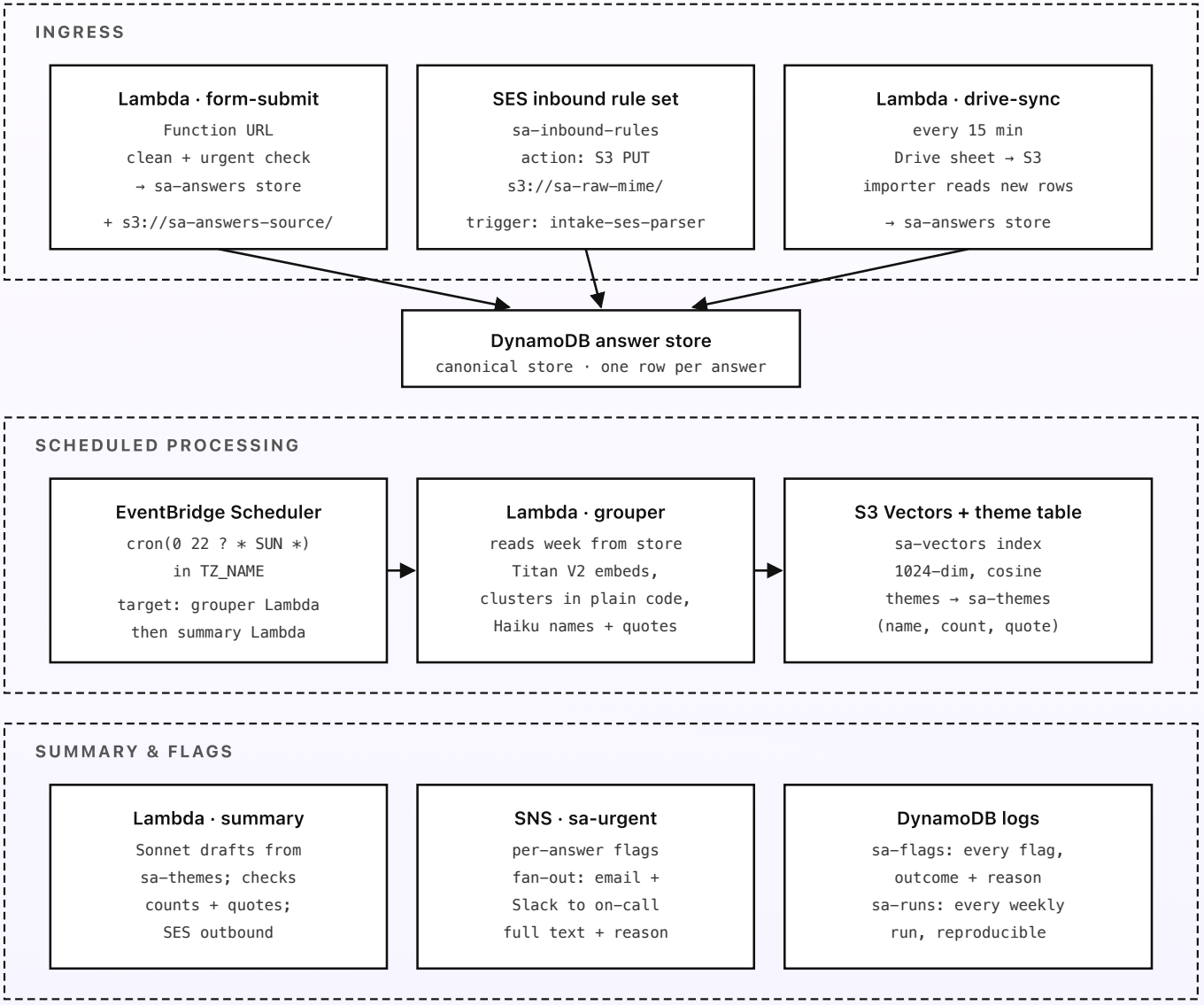
Engineering reference: the survey analyzer architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, the S3 Vectors index, Lambda inventory, IAM scopes, EventBridge Scheduler config, the DynamoDB schemas, and the SNS fan-out. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). SES inbound, Bedrock Global cross-Region inference, S3 Vectors, and EventBridge Scheduler are all in good shape there. A second region for multi-region resilience isn't worth the extra setup work at SMB volume — the failure mode for an SMB is a weekly summary that goes out a day late, not a regional outage. One AWS account dedicated to the analyzer (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Every answer is embedded and urgent-checked — and every run and flag is logged.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the answer store), scheduled processing (the weekly grouper building the theme table), summary and flags (the summary ships and per-answer urgents fan out via SNS). Every Lambda is event- or schedule-driven; nothing is synchronous-chained.

Lambda functions

All Lambdas use the `arm64` architecture, the smallest memory size that meets latency targets (typically 256 MB), Python 3.14 runtime, and CloudWatch Logs at 7-day retention. Each function has its own least-privilege IAM role. None run inside a VPC.

- **`form-submit`** — Lambda Function URL, `AuthType: NONE`; verifies a shared secret (HMAC over the body) the survey form includes. Cleans the answer text, runs the urgent check via Bedrock Haiku 4.5, writes the answer to `sa-answers`, and mirrors the raw payload to `s3://sa-answers-source/`. On an urgent or callback verdict, publishes to `sa-urgent` or appends to the callback queue. Memory: 256 MB. Timeout: 15 s.
- **`intake-ses-parser`** — S3 PUT trigger on `s3://sa-raw-mime/`. Parses MIME, strips signatures and quoted reply trails, and extracts one or more answers from the body or from a CSV/XLSX attachment (`openpyxl` for spreadsheets). For genuinely ambiguous layouts only, calls Bedrock Haiku 4.5 to split the text into discrete answers. Each extracted answer runs the same clean + urgent-check + store path as `form-submit`. Memory: 512 MB. Timeout: 60 s.
- **`drive-sync`** — EventBridge Scheduler target, fires every 15 minutes. Uses the Google Drive API + Sheets API (service-account credentials in Secrets Manager

under `sa/drive/sa`) to export the answer sheet as CSV and write to `s3://sa-answers-source/answers.csv` only if the sheet has changed since the last sync. A small importer step reads rows not yet seen and runs them through the clean + urgent-check + store path. Same pattern syncs the rules and voice docs to `s3://sa-rules-source/`. Memory: 256 MB. Timeout: 30 s.

- **`grouper`** — EventBridge Scheduler target, weekly (Sunday 10pm local in `TZ_NAME`). Reads the period's answers from `sa-answers`. Calls Bedrock Titan Text Embeddings V2 (`amazon.titan-embed-text-v2:0`, 1024-dim) for each new answer and upserts into the `sa-vectors` S3 Vectors index. Pulls the vectors back, clusters with HDBSCAN in `scikit-learn` / `hdbscan`, drops clusters below `min_theme_size`, then calls Bedrock Haiku 4.5 (`global.anthropic.claude-haiku-4-5-20251001-v1:0`) once per surviving cluster to name it and confirm a representative quote (the answer nearest the cluster centroid). Writes the theme table to `sa-themes`. Memory: 1024 MB. Timeout: 300 s.
- **`summary`** — EventBridge Scheduler target, weekly just after `grouper` (chained via the Scheduler flexible time window, or invoked at the tail of `grouper`). Reads `sa-themes` and the week's `sa-flags`; calls Bedrock Sonnet 4.6 (`global.anthropic.claude-sonnet-4-6-20250930-v1:0`) to draft the summary; runs the count check (every number against the cluster sizes) and the quote check (every quote against `sa-answers`); shapes to the voice doc and length cap; sends via SES `SendRawEmail`. Writes the run to `sa-runs`. Memory: 512 MB. Timeout: 120 s.
- **`callback-sweep`** — EventBridge Scheduler target, daily at 9am local. Reads the callback queue accumulated by intake, posts a single digest to the on-call

Slack channel, and clears the entries it lists. No Bedrock. Memory: 256 MB. Timeout: 30 s.

Storage

- **DynamoDB** · `sa-answers` — one row per answer. PK `answer_id`; attributes: `survey`, `received_at`, `rating`, `raw_text`, `clean_text`, `urgent` (bool), `theme_id` (set by the weekly grouper). GSI on `received_at` for the weekly range read. On-demand.
- **DynamoDB** · `sa-themes` — one row per theme per weekly run. PK `(run_id, theme_id)`; attributes: `name`, `count`, `quote_answer_id`, `centroid_ref`. On-demand.
- **DynamoDB** · `sa-flags` — one row per urgent-check outcome. PK `(answer_id, ts)`; attributes: `outcome` (urgent/callback/normal), `reason`, `notified`. On-demand. No TTL — this is the long-term flag trail.
- **DynamoDB** · `sa-runs` — one row per weekly run. PK `run_id`; attributes: `window_start`, `window_end`, `n_answers`, `n_themes`, `draft`, `final`, `checks_passed`. On-demand. Keeps each summary reproducible.
- **S3 Vectors** · `sa-vectors` — the answer embeddings. 1024-dim, cosine distance, metadata `{answer_id, received_at}`. Queried for clustering and centroid lookup. No always-on cluster.
- **S3** · `sa-answers-source` — mirrored CSV from the Drive sheet and raw form payloads. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `sa-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.

- **S3 · `sa-raw-mime`** — raw inbound MIME from forwarded feedback. Lifecycle to Glacier at 30 days; expiry at 7 years.

Bedrock

- **Embeddings.** `amazon.titan-embed-text-v2:0`, 1024 dimensions, normalized. One call per answer in `grouper`; results stored in the `sa-vectors` S3 Vectors index.
- **Cheap-path model.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0`. Callsites: the per-answer urgent check (in `form-submit` and `intake-ses-parser`), the optional answer-splitting in the parser, and the per-theme naming in `grouper`.
- **Heavier model.** `anthropic.claude-sonnet-4-6-20250930-v1:0` via `global.anthropic.claude-sonnet-4-6-20250930-v1:0`. One callsite: the weekly `summary` draft, where weighing several themes and striking a tone justifies the heavier model. Fires once a week.
- **Quotas.** Default account quotas are more than enough at SMB volume. The per-answer Haiku check is the highest-frequency callsite; it's a short prompt with a one-token verdict.

EventBridge Scheduler config

- **`sa-weekly-group`** — `cron(0 22 ? * SUN *)` in the SMB's timezone. Target: `grouper` Lambda.

- `sa-weekly-summary` — `cron(30 22 ? * SUN *)` in TZ (30 minutes after the grouper). Target: `summary` Lambda.
- `sa-drive-sync` — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- `sa-callback-sweep` — `cron(0 9 * * ? *)` in TZ. Target: `callback-sweep` Lambda.

SES inbound/outbound and SNS

- Set the MX record on a dedicated subdomain (e.g. `feedback.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `sa-inbound-rules`: one rule with recipient `feedback@your-company.com` → spam scan → S3 PUT to `s3://sa-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-ses-parser`.
- SES outbound for the weekly summary: verify a sender identity at `insights@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request.
- SNS topic `sa-urgent`: subscriptions for the on-call email and a Slack channel (via an AWS Chatbot configuration or a small relay Lambda). This is the path an angry answer takes within a minute of landing.

IAM (least privilege per Lambda)

Each Lambda has its own role with policies scoped to exact ARNs. Sketch:

- **form-submit role:** `dynamodb:PutItem` on `sa-answers` and `sa-flags`; `s3:PutObject` on `sa-answers-source`; `bedrock:InvokeModel` on the Haiku

ARN; `sns:Publish` on `sa-urgent`; `secretsmanager:GetSecretValue` on the shared-secret.

- **grouper role:** `dynamodb:Query` on `sa-answers` (the `received_at` GSI) + `PutItem` on `sa-themes`; `bedrock:InvokeModel` on the Titan and Haiku ARNs; `s3vectors:PutVectors` + `QueryVectors` on the `sa-vectors` index. *No* SES, *no* Sonnet.
- **summary role:** `dynamodb:Query` on `sa-themes`, `sa-flags`, and `sa-answers` (for the quote check) + `PutItem` on `sa-runs`; `bedrock:InvokeModel` on the Sonnet ARN; `ses:SendRawEmail` from the verified sender identity.
- **intake-ses-parser role:** `s3:GetObject` on `sa-raw-mime`; `dynamodb:PutItem` on `sa-answers` and `sa-flags`; `bedrock:InvokeModel` on the Haiku ARN; `sns:Publish` on `sa-urgent`.
- **drive-sync role:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the answers and rules buckets; `dynamodb:PutItem` on `sa-answers`; outbound network to `www.googleapis.com`.

Urgent check and clustering internals

The urgent check is a single Haiku call with a JSON-only contract: `{"verdict": "urgent|callback|normal", "reason": "<one line>"}`. The system prompt embeds the rules-doc definition of urgent and forbids any action beyond classification — the model never drafts a reply. A verdict of `urgent` triggers an `sns:Publish`; `callback` appends to the callback queue; `normal` stores and moves on. The verdict is written to `sa-flags` in all three cases.

Clustering uses HDBSCAN over the 1024-dim vectors, which finds dense groups without forcing a fixed cluster count and labels sparse points as noise (the long tail). `min_cluster_size` is derived from `min_theme_size` in the rules doc. The count for a theme is `len(cluster_members)` — a plain integer, never model-generated. The representative quote is the member with the smallest cosine distance to the cluster centroid, resolved back to its `answer_id` and pulled verbatim from `sa-answers`.

Observability and cost gates

- **CloudWatch Logs:** all Lambdas, 7-day retention, structured JSON. Subscription filter on `"error"` + `"throttle"` + `"timeout"` to a CloudWatch metric for alerting.
- **Alarms:** grouper or summary failures > 0 in a week (the weekly pass is the one piece that has to run); urgent-publish failures > 0 (a missed angry customer is the costly miss); count-check or quote-check rejections > 2 in a run (might mean a prompt regression).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $30/month threshold, alarm at 80% and 100%, posts to SNS topic `sa-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for the Drive and Sheets APIs live in Secrets Manager under `sa/drive/sa`. The form-submit shared secret lives under `sa/form/secret`; the Slack relay token (if used) under `sa/slack/*`. SES sender

identity lives in IAM and the verified-domain config. The configured timezone, the urgent definition reference, `min_theme_size`, the summary length cap, and the on-call and owner addresses all live in Parameter Store under `/sa/config/`. Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC into a deploy role (no long-lived keys) running AWS SAM. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), create the S3 Vectors index in its own stack so a re-index doesn't churn the rest, turn on S3 versioning for `sa-answers-source` and `sa-rules-source` so a bad Drive paste can be rolled back in one click, and pin the EventBridge Scheduler timezone so you don't accidentally run the weekly pass in UTC after a CI rotation. Total deployable surface: around six Lambdas, four DDB tables, one S3 Vectors index, three S3 buckets, the Scheduler rules, one SES rule set, two SNS topics, and one Budgets alarm.

That's the full system. Six narrative posts and this engineering reference. If you want to talk about adapting it for your business, see Work with me.